Copyright

by

Alexander Eric Braylan

2019

The Thesis Committee for Alexander Eric Braylan
certifies that this is the approved version of the following thesis:

Object-Model Transfer in the General Video Game Domain

Committee:

Risto Miikkulainen, Supervisor

Peter Stone

Object-Model Transfer in the General Video Game Domain

by

Alexander Eric Braylan

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

The University of Texas at Austin

August 2019

Object-Model Transfer in the General Video Game Domain

Alexander Eric Braylan, MSCompSci
The University of Texas at Austin, 2019

Supervisor: Risto Miikkulainen

Reinforcement learning agents often benefit from learning models that predict their environment. However, learned models may not generalize well to novel situations. This thesis investigates the potential for a *transfer learning* approach to address the challenge in the video game domain. The approach helps agents learn models of new games by transferring knowledge from previously learned games. Transfer is facilitated by decomposing games into the objects they contain. The assumption is that it is easier to relate features between objects from different games than features between whole environments of different games. Experiments show that predictions made by this method are more accurate than predictions made without transferred knowledge, and this improvement is demonstrated to result in increased efficiency in a task where an agent explores a maze-like game. The conclusion is that model learning can be enhanced by transferring object models from previously learned environments.

Contents

Chapter 1

Introduction

Reinforcement learning agents for playing video games have achieved high levels of performance across a broad spectrum of games, but it is unclear how well these agents would be able to generalize to unfamiliar experiences. In sequential decision problems such as video games, it is particularly desirable for agents to be able to use what they learn from their experiences in a task to then perform reasonably well in a qualitatively similar task [36]. In video games, this should mean being able to play subsequent levels of a game after mastering a first level. It should also mean being able to play a flipped or rotated version of a mastered game. With some stretch it should also mean being faster to learn unfamiliar games after mastering other similar ones. This thesis explores some methods for agents to start learning to play in an unfamiliar environment after making only very few observations in it.

The first general idea is to learn *forward models* of an agent's environment, in order to reduce the amount of required training and improve overall performance and flexibility. A forward model is a function that predicts the future state of an environment from its current state. In problems with sparse rewards, learning a model takes advantage of supervised learning when visiting states that are far away from reward-giving states. To help understand how a forward model can reduce the amount of required training, consider how a

simple model-free value-learning agent might tackle a game where an agent moves left and right on a one-dimensional line trying to reach the opposite end. The value-learning agent must first reach the goal to learn the value of the neighboring states. It must then gradually assign values to the states further from the goal. If the agent starts at the left end, and the goal is at the right end, it must go to the goal and then back to learn how to reach the right end from the left end. In contrast, a model-learning agent remembers that each state can lead to its neighboring state as the line is traversed. At the moment it reaches the goal, it has sufficient information to plan a trajectory from the left end to the right end. Additionally, it can use its knowledge to inform the exploration that gets it to the goal in the first place.

Another advantage of using model learning for reinforcement learning problems is that it becomes easier to partition experiences into training and test data. The reason is that model accuracy can be the metric of interest instead of cumulative reward. For example, in the video game domain, if the metric of interest is cumulative reward, an entire episode must be put into either the training or testing data set. For deterministic games, a problem arises because both data sets contain many similar episodes, with states encountered and learned during training appearing again in test episodes. On the other hand, if the metric of interest is model accuracy, individual transitions from the same episode can be split between training and testing data. The ability to split data from a single episode between testing and training enables more meaningful assessment of agents' generalization ability in deterministic games. If the smallest unit of data is a transition, even when the game is deterministic, transitions from different parts of the same episode can be divided between sets so as to avoid the overlapping of identical data.

One challenge with model learning is that when the data set used to train a model is small, noisy, or high-dimensional, the model is at higher risk of suffering from *generalization error* in predictions made outside of the data seen during training. An example of the small data problem is when the first few frames of a new game are not sufficient to inform the agent about how the game will behave later on. An example of the noise problem is

when random sensor error causes a complex model to over-fit the data. An example of the dimensionality problem is when too many variables are allowed to be conditionally dependent, so an agent learns "superstitious" relationships between variables that are coincidental in the observed data. The severity of each challenge is relative to the others – for example, if the quantity of the data is large enough, high dimensionality may not be an issue, and so on. However, problems with generalization error often involve all three types of adversity.

One field of research that may help address the problem of generalization error is *transfer learning* [34; 28], the reuse of knowledge and skills learned in *source* tasks to accelerate and improve performance in a different *target* task. Applied to video games, the idea is that an agent with ample experience playing various source games can learn a better model of a new target game by transferring and combining knowledge from the source games. This thesis considers the role of this combined transferred knowledge in forming an *inductive bias* — an assumption that constrains the space of possible models, and a way to guard against generalization error [25].

The first question in developing a transfer learning method is how to map between variables in the source and target environments. A second question is how to integrate and apply the transferred knowledge. This thesis responds to both questions in the context of general video game playing. The key step taken toward the first challenge is to decompose game environments into collections of objects. In an object-oriented formulation of a game environment, objects belong to *object classes* exhibiting similar behaviors across a wide variety of games [14]. The variables of an object class are interpreted in the same way regardless of the game, simplifying the question of variable mapping. The approach of this thesis toward the second challenge is to construct *transfer ensembles* out of models transferred from source games and *scratch* models newly trained for the target game. Each transfer ensemble uses a weighted average of predictions from its constituent models to predict the behavior of a target object. The weights are calculated based on how well each constituent model describes the data observed for the target object class.

Experimental results show that agents using transfer ensembles as models of object classes generalize better than using scratch models. After observing small quantities of *in-sample* training data, transfer ensembles achieve greater accuracy than scratch models when predicting the behavior of objects in subsequent *out-of-sample* test data. Transfer ensembles also generalize better than scratch models when random noise is artificially added to the training data. Agents that use learned models to inform their actions in an exploration task are shown to perform better when using the transfer learning approach than when learning from scratch.

Altogether, the conclusion is that decomposing environments into objects and transferring object models across games is a promising approach for general video game playing and other multi-task sequential decision problems.

Chapter 2

Background

This thesis draws from research in general video game playing, model-based reinforcement learning, and transfer learning. Each is a broad field of research, so this section will review the topics most relevant to this work.

2.1 General Video Game AI

General Video Game AI (GVG-AI) is an open-source project that facilitates artificial intelligence research in general video game playing [30]. The GVG-AI project provides a framework for agents to interact with games and includes 60 games hand-coded in the Video Game Description Language [16]. The games are similar to games from the Atari 2600 console and other popular video games, including games inspired by Space Invaders, Frogger, Zelda, Lemmings, Seaquest, Sokoban, Dig Dug, Pacman, Star Fox, Plants vs. Zombies, among many others. Borrowing from several genres of video games presents agents with a wide diversity of challenges.

Additionally, there are a few advantages to using GVG-AI over an Atari emulator. The GVG-AI code can display exact information about game objects, while for Atari this information must be extracted [7; 18]. Furthermore, GVG-AI objects can exhibit stochastic

behavior. For Atari, stochasticity can so far only be added artificially to the initial game state or to the actions input by the player [19]. Finally, each game in GVG-AI includes several levels with different initial conditions, allowing for straightforward out-of-sample testing. For these reasons, the experiments in this thesis use the GVG-AI framework and games.

2.2 Model-Based Reinforcement Learning

Reinforcement learning problems challenge agents to take actions in response to observations of an environment in order to accumulate rewards over time [32]. In the most common case, the environment is formally a Markov decision process (MDP), which consists of a set of states, actions, and a probabilistic transition function. This function governs the distribution of subsequent states given every current state and action.

2.2.1 The Benefits and Challenges of Models

Model-based reinforcement learning methods rely on an estimate of the transition function. Compared to *model-free* methods, they use richer representations of the environmental dynamics. Such representations yield various benefits: data efficiency, better planning and exploration, and robustness against changes in the reward function [3; 2].

When a model must be learned from observation data, the model-based approach must deal with the "curse of dimensionality" [8]. The number of states grows exponentially with the number of state variables, making it intractable to maintain a full table of observed transition probabilities. Therefore, most approaches to model learning for high-dimensional environments use *factored state* representations, learning approximate transition functions on a manageable number of features of the state space with the assumption that the choice of features allows the learned function to generalize across experiences.

2.2.2 Factored-State Model Learning for Video Games

Because video games are high-dimensional environments, the only approaches that learn models of the environment use factored state representations. An approach to learning models of Atari games by Bellemare et al. [6] predicts patches of pixels from neighboring patches using a compression algorithm, taking advantage of the tendency of game objects to depend only on nearby objects. Alternatively, a deep learning approach by Oh et al. [27] uses a convolutional neural network on pixel inputs from the whole game screen to predict future pixel values.

Oh et al. also report interesting results on how useful the models learned by their methods are. First of all, they note how small differences in the mean squared error accuracy metric used to evaluate the learned models correspond to large differences in scores achieved by the agents using the models. They reckon that a large quality difference in the model of the player-controlled object accounts for a significant performance difference but gets obscured by the global accuracy metric that weighs all regions and objects of the game similarly.

While some research exists on learning factored models to make the most out of few training samples [13; 21; 22], both papers on model learning for video games focus on the scalability and power of the models rather than on sample efficiency. Nevertheless, the differences between the two approaches have important implications for the question of how well they generalize after seeing few training samples. Allowing the model to capture relationships between any pixels on the screen results in a better fit to the data than only modeling relationships between neighboring patches, but requires larger amounts of data to avoid overfitting. The neural networks used by Oh et al. trained on 500,000 frames per game. The more constrained model based on neighboring patches in Bellemare et al. could possibly have more generalization power in a sparse training scenario, but the reported results come after 10 million frames of training per game.

2.2.3 Object-Oriented Markov Decision Process

An *Object-Oriented Markov Decision Process* (OO-MDP) is a factorization that exploits the object-oriented nature of many reinforcement learning problems by re-framing the environment as a collection of objects [14]. Compared to the high dimensionality of the full game state, the object-oriented environment is represented only by the relatively few *attributes* of each object. These attributes include the object's internal state variables and variables representing relations with other objects. For example, geographic relationships are encoded by first-order propositions $on(o_1, o_2), touch^N(o_1, o_2), touch^E(o_1, o_2), etc.$ Each object belongs to an *object class*; all instances of the same object class are assumed to follow the same transition function, thus only one model is needed for each object class. The assumption that many of these object classes are similar over multiple games is one motivation for choosing the object-oriented factorization for transfer learning.

While GVG-AI provides knowledge of object class and location to agents, some other video game domains may not offer an object-oriented representation to be used directly, but only the raw pixels or RAM memory state. This thesis makes the assumption that agents are able to detect the locations of objects, as has been done in various work across multiple domains. In the Atari domain, several methods exist for detecting and classifying game objects, and some game-playing agents use the extracted object representations as their inputs [7; 18]. Several competitions for the game Ms. Pacman have also provided agents with object location information [1; 29]. The Unreal Tournament BotPrize competition allows bots to use inputs sensing the relative locations of objects such as enemies, items, and ledges among other important variables [31]. Practical applications of artificial intelligence in commercial video games, like such bots used in first-person shooters, rarely or never rely on learning from raw pixels or RAM state. The intelligent creatures in the game Black and White, famous among games for its use of learning agents, represents its context using attributes linked to objects and characters [24]. The necessity of object-oriented representations for a model learning approach seems to scale with the complexity

of the game.

2.3 Transfer Learning

There are many ways to transfer knowledge between tasks, including policy transfer, value function transfer, and model transfer. Transfer of policies between video games was examined by Braylan et al. [10], showing signs that policy transfer can potentially improve training efficiency and result in higher-scoring agents. However, these policies are generally complex functions such as neural networks with as many input variables as locations on the whole grid times the number of possible object classes, making it difficult to understand what aspect of a policy gets transferred. Models of individual objects in video games can be much less complex, and it may be easier to see how they transfer.

While there has been prior work on transfer learning for model-based reinforcement learning [33], the potential for applying transfer learning to the model learning problem is not addressed in any prior work on general video game playing. In the approaches of Bellemare et al., Oh et al., and Braylan et al., the environment variables modeled are raw pixels or locations of objects across the whole grid, which are difficult to relate between games without any transformation. Models of objects can have fewer variables, which are also comparable across games, so it should be easier to understand how they transfer.

The transfer of models between tasks is related to the theory behind choosing a good inductive bias. When a learner can sample from multiple related tasks, an inductive bias that works well for several of those tasks can be expected to work well in other related tasks [5]. For example, learning multiple related tasks at the same time with some shared learning parameters can be better than learning each task individually [11]. Similarly, source knowledge can inform the selection of inductive bias in target tasks. Some such approaches involve the use of an ensemble model, a weighted combination of source models where the weights depend on how well each source model predicts the target data [12; 17; 15; 4]. This thesis takes this type of approach, with an added step of further adapting

the ensemble to target data by retraining the constituents' parameters.

Chapter 3

Approach

This chapter first presents a general overview of the approach as it applies to sequential decision-making domains, while later sections describe how this approach works specifically for GVG-AI games, on which the experiments in this thesis are conducted. Included in the approach are a method for learning object models from scratch in GVG-AI games and a transfer learning method for reusing scratch models to create better models in target games.

3.1 General Process

The approach presented in this thesis should be applicable to a wider range of domains than just GVG-AI. This section outlines the necessary steps that must be taken in order to learn and transfer object class models in general. It should serve as a guideline for judging when the methods may be inappropriate or may need tailoring in order to apply them to other domains.

3.1.1 Object-oriented Representation of the Game State

First, the inputs available to the agent must be sufficient to create an object-oriented representation. This representation should include estimates of the locations and classes of other objects in the game. It could also include estimates of global game variables such as score and time remaining or other object variables such as health and ammo. Dividing the game state into object states should reduce the total number of variable interactions modeled. Each object maintains its own variables, chosen as described below. Variables representing relations between multiple objects or with global variables can be maintained by one or multiple of those objects. Some error in the estimated object representation is permissible.

3.1.2 Determination of Object Class Attributes

Some knowledge of the domain is necessary to determine the attributes used as variables for object models. These attributes should include variables that are important for the game objective and variables that are important for predicting the state of the object. It is permissible to include unimportant variables, but using more variables requires training on more data in order to generalize well. It is also desirable for object variables to be chosen so as to map between different object classes as intuitively as possible, in order to facilitate transfer learning. For example, directional movement and orientation variables are useful because they are qualitatively similar across object classes and can help predict subsequent object states.

3.1.3 Learning Object Class Models

A forward model is trained for each object class by fitting it to observed data from all object instances of that class. A forward model F_j of an object class j generates a prediction $\hat{S}_t^i = F_j(S_{t-1}^i)$ for the state S_t^i of object instance i (belonging to object class j), given its previous state S_{t-1}^i. It should be possible to factor the state into variables, which include global variables such as the player action. The forward models of objects are used to predict

their future states, which should be useful for the greater tasks carried out by the agent such as exploring and staying alive.

3.1.4 Transfer of Object Class Models Between Tasks

A domain may consist of multiple tasks, such as the multiple games in the video game domain. Objects appearing in one task may be similar to objects that have appeared in other tasks, and therefore their future states may be predictable using the same object class model. When it is difficult to model objects using data from an unfamiliar task, object class models from familiar tasks can be transferred in order to improve the generalization ability of the new model. Data from the unfamiliar task is used to calculate a measure of how likely a new object is to belong to a known object class. Part of that measure should be the accuracy of predictions made by the transition function model for the candidate class on the new object data. An ensemble of candidate models is used to predict the new object, with the weight of each ensemble being informed by the likelihood measure. After the ensemble is composed, it is further tuned to fit the data observed in the unfamiliar task.

3.2 Interfacing with GVG-AI

The GVG-AI project supplies a free and self-contained Java code repository (gvgai) designed to easily run GVG-AI games so that either a human or an artificial agent may play them. The version of the gvgai repository used in this thesis is for the 2015 competition last updated on September 27, 2015.

An artificial agent can interact with games by receiving observations of the game state and outputting an action each frame. The artificial agent must define an extension of GVG-AI's `AbstractPlayer.java`, implementing the method `Types.ACTIONS act(StateObservation stateObs, ElapsedCpuTimer elapsedTimer)`. This method takes as input a `StateObservation` and outputs a `Types.ACTIONS` action. The `StateObservation` contains a list of all the game object instances. For each of

these objects, it shows the x and y position as well as an integer representing the object's class. This number is useful for grouping different instances of the same class within a game. For example, in the game Aliens, all instances of wall objects are denoted with class "0", the player's avatar with class "1", and so on. An example game with its underlying object class information is depicted in Figure 3.1.

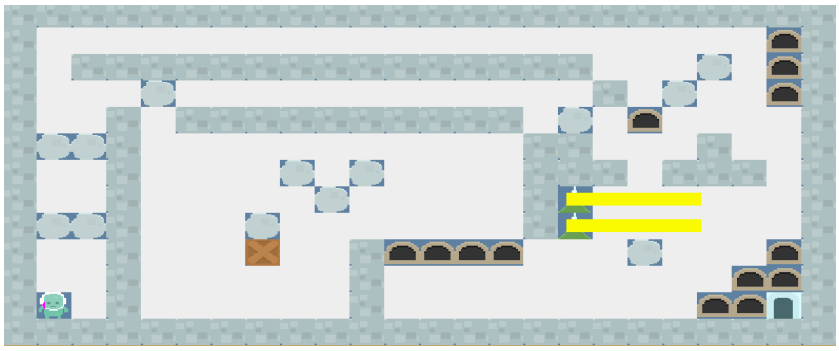
The position and object class information are sufficient to extract a set of attributes that capture most of the observable object behaviors in GVG-AI games. The most common behaviors encountered include deterministic, stochastic, and player-controlled movement; death; and spawning of other objects on the same tile. Spawning is a novel extension of the OO-MDP formulation to capture the effect of a new object instance appearing in a game. In GVG-AI games, after the first frame, new object instances only appear by being spawned by other objects. There are no games where a new object instance appears in an area of the game where there are no other objects. Therefore, the appearance of new objects can be modeled as a behavior of the spawning object. In other domains, spawning may need to be handled differently, for instance by using dummy objects to represent spawn points.

3.3 Model Variables: Object Class Attributes

Object attributes are selected manually based on observations from 30 GVG-AI games. These 30 games are withheld from the experiments in order to avoid contamination, so that the results can be expected to hold in other domains. Choosing object attributes is a form of manual transfer learning, in that observations from the withheld games are the source for features and assumptions in the approach used for the remaining target games. How to automate this step is an interesting challenge for future work.

The object attributes chosen include variables used as inputs and outputs to the object class models, with inputs predicting outputs. The predicted next state of an object instance g consists of the following *output* attributes:

- Directional movement (North/South/East/West) at time t, $D_t = \{d_t^N, d_t^S, d_t^E, d_t^W\}$;



(a) Boloadventures screenshot

0																						2	0
0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				6		2	0
0				6													0		6			2	0
0			0		0	0	0	0	0	0	0	0	0	0		6		2					0
0	6	6	0												0	0				0			0
0			0					6		6					0	0	0		0	0	0		0
0			0						6						0	16	16	16	16				0
0	6	6	0				6								0	16	16	16	16				0
0			0				4			0	2	2	2	2				6				2	0
0			0							0											2	2	0
0	1		0							0										2	2	3	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b) Boloadventures object locations and classes

Figure 3.1: Example game screen and extracted object information. Location and class information is extracted at each frame for each object instance present. This data is sufficient for building object class models.

- Whether the object is alive and on screen, e_t; and
- New spawns of other objects on the tile of this object, $N_t = \{n_t^{C(i)} : \text{spawn}_t(i), \text{on}_t(g, i)\}$.

To clarify how the spawn attributes are managed, the proposition $\text{spawn}_t(i)$ denotes whether an object i is a *spawn* – a newly observed object instance in a game – at time t. Every spawn observation is recorded as $n_t^{C(i)} = 1$ for every other object on the same tile as the spawn, with $C(i)$ denoting the object class of object i. For example, when a new bomb object appears on the same tile as an alien object, that alien object takes a value of 1 for the attribute n^{bomb}.

In addition to the above output attributes, the following *input* attributes account for factors upon which the predicted behaviors are conditioned:

- Directional movement at time $t-1$, $D_{t-1} = \{d^N_{t-1}, d^S_{t-1}, d^E_{t-1}, d^W_{t-1}\}$;
- Whether the object was *facing forward*, f_{t-1}, satisfied when the last two movement actions were in the same direction;
- Other objects touching the object $H_{t-1} = \{$
  $h^{N,C(i)}_{t-1}$: $\text{touch}^N_{t-1}(g, i)$,
  $h^{S,C(i)}_{t-1}$: $\text{touch}^S_{t-1}(g, i)$,
  $h^{E,C(i)}_{t-1}$: $\text{touch}^E_{t-1}(g, i)$,
  $h^{W,C(i)}_{t-1}$: $\text{touch}^W_{t-1}(g, i)$,
  $h^{O,C(i)}_{t-1}$: $\text{on}_{t-1}(g, i)$
  $\}$; and
- Action input by the player, $A_{t-1} = \{a^{\text{NIL}}_{t-1}, a^{\text{UP}}_{t-1}, a^{\text{DOWN}}_{t-1}, a^{\text{LEFT}}_{t-1}, a^{\text{RIGHT}}_{t-1}, a^{\text{SHOOT}}_{t-1}\}$.

Whenever an object instance is adjacent or overlapping another object instance of a different class, its h attribute corresponding to the other object's class and relative position takes a value of 1.

In addition to object class models, termination models can be learned for predicting whether the game is won or lost at each frame from some global game variables. Termination models are not deeply explored in this thesis but are helpful for experiments involving action selection. The two termination models, $P(\text{WIN}|X)$ and $P(\text{LOSE}|X)$, are conditioned on the following inputs:

- Existence of at least one live object instance of each class in the game,
  $X_{t-1} = \{x^j_{t-1} : \text{exists}_{t-1}(j)\}$.

Each x^j represents whether any instance of the game's object class j exists at all at the given time. These inputs are used because in most GVG-AI games, termination coincides with the total disappearance of one of the game's object classes.

3.4 Data Collection

Object models are learned after collecting data by playing games. During one game episode, at every frame a call is made to the agent's `act` method. Inside this method, the model learner extracts the necessary data from the received `StateObservation` and returns an action. At each frame, each object instance in the game contributes a data point to the model for that object class.

3.5 Functional Form and Learning Algorithm of Scratch Model

The state space for an object is small enough that its observed transition probabilities can be held in a table. Nevertheless, a factored state representation ensures that the system is able to scale up to more complex object models if necessary. Furthermore, a factored model can generalize better than a lookup table. To see why, consider observing the avatar dying every time it collided with a bomb. A factored model would associate the input variable representing the collision with a bomb to the output variable representing death. A lookup table, on the other hand, would associate the entire state, including all other input variables unrelated to the collision with the bomb. If the agent were to encounter a state where a collision with a bomb was imminent but another variable not seen in training were also activated, such as bordering some other object like a coin or a wall, the lookup table would not know to predict death. The lookup table would have to perform some additional interpolative reasoning such as finding the closest known state, whereas the factored model by default receives little influence from unrecognized values of state variables.

In a factored state model, the prediction $\hat{S}_t$ of the next state of an object (conditioned on an action) is decomposed into predictions for each output variable $\hat{s}_t^k \in \hat{S}_t$. All of the specified object variables take values of either 0 or 1. The values of variables not observed by an object are 0 by default. The factored-state model produces a prediction between 0 and 1 for each output variable of an object instance. Each prediction represents the probability

of the output variable taking a value of 1 given the observed input values. The model is trained for each output variable of each object class using observations of all instances of the object class in a game. In other words, the factored state model learns the object's transition function from the history of state inputs and outputs and actions.

Common forms for factored state forward models include logistic regressions and neural networks. A logistic regression output is a sigmoidal function of its weighted inputs, taking a form such as $\hat{s}_t^k \sim \frac{1}{1+e^{-WS_{t-1}}}$. The weight vector W consists of coefficients to the input variables and an intercept term which are trained through gradient descent. The gradient descent algorithm iteratively decreases a cost computed from the values of observed S_t and predicted $\hat{S}_t$ by the logistic regression until it converges near a global minimum [9]. An appropriate cost function for a model with binary outputs is the cross entropy error $E_t = -\sum_k (s_t^k \ln \hat{s}_t^k + (1 - s_t^k) \ln (1 - \hat{s}_t^k))$. Weights are gradually changed in the direction of the partial derivative of this cost with respect to the weights so as to reduce the cost; the partial derivative for a weight between input i and output j is $(\hat{s}_t^j - s_t^j)s_{t-1}^i$. During gradient descent training, data points are presented in random order at each iteration to avoid biasing the learned model.

A neural network is a multi-layered model whose nodes may also be sigmoidal functions of the weighted inputs, in which case a neural network is comparable to a hierarchical logistic regression. Neural networks have the power to capture complex non-linear relationships in the data, but their training algorithms are only guaranteed to converge at local, not global, minima. Being more complex, neural networks are also more prone to overfitting than logistic regression [20].

The object models used in this thesis take the form of logistic regression. Some of the more complex behaviors in certain game objects are therefore not possible to capture perfectly with these models. However, since a major objective is to address the problem of learning models that can generalize well from sparse training data, the simplicity and global convergence of logistic regression models are preferred over the fitting power of

neural networks.

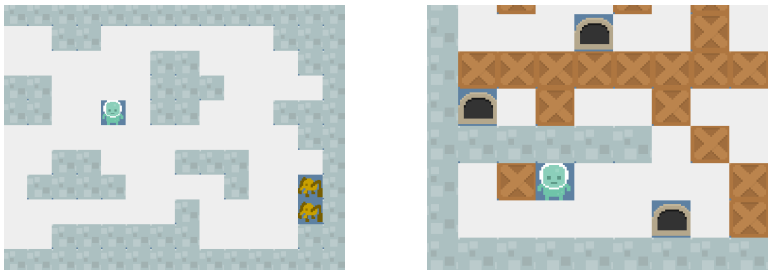
3.6 Object Model Transfer

The transfer learning approach in this thesis relies on a simple and intuitive assumption: Some object classes encountered in a target should behave similarly to other object classes previously encountered in sources. Therefore, when reasoning about an unknown target object, knowledge of previously seen similar source objects can help constrain and shape the distribution of predictions for the target object's behavior. The measure of similarity depends on what is known about the target object, what is known about the source objects, and the ability to establish relationships between the attributes of different objects. This assumption forms an inductive bias which should help trained models generalize better to unseen target data. The following example serves to illustrate more tangibly how object class models can be transferred.

3.6.1 An Illustrative Example of Walls and Avatars

In the game Chase, the player-controlled avatar must chase goats by moving freely in four directions except when blocked by wall objects. These movement rules for the avatar are common in several other games, such as the game Escape. In Escape, the avatar is again moved in four directions by the player and is blocked by walls. Additionally, the Escape avatar can push away box objects and disappear through hole objects. A transfer learning agent who has played many games of Chase but has only seen a few frames of Escape should be able to reuse specific knowledge from Chase to make more accurate predictions about Escape than a total novice.

Upon encountering a wall for the first time in Escape, a novice agent with no Chase experience would have low certainty on the outcome of an attempt to move the avatar into the wall. In contrast, a transfer learning agent could notice some similar behavior between the Chase and Escape avatars — such as how the player inputs move both of them in similar



(a) Chase screenshot (b) Escape screenshot

Figure 3.2: Chase and Escape example frames. Several object classes exhibit similar behaviors across games, such as the little green avatar and the rectangular gray walls that appear in both Chase and Escape. Transfer is most useful across games that share similar object classes.

directions — and reason that the interaction with the wall is also likely to be the same in both games. Likewise, a novice agent whose North side has always been blocked by walls in its limited experience would not realize that it could move up once the wall disappeared, while a transfer learning agent could expect that ability.

The transfer learning method presented in this thesis produces models that make predictions as described above when the source is Chase and the target is Escape. However, transferring object class models is not always so simple for all sources and targets. The following four subsections explain additional challenges encountered and how they are addressed.

3.6.2 Source Knowledge Selection

One objective for a transfer learning algorithm is an ability to choose automatically what knowledge to transfer from a potentially large pool of source models. Transferred knowledge may harm rather than improve performance in the target task, an outcome called *negative transfer* [34]. In order to avoid negative transfer, the transfer learning algorithm may employ a method for estimating the expected improvement in performance due to transfer.

This thesis adopts a measure of one-frame forward prediction accuracy to evaluate

learned models, both for guiding source selection and for overall evaluation. Prediction accuracy of an object class model F on object transition data $\mathbb{S} = \{S_1, S_2, ...S_T\}$ is calculated as accuracy$(F, \mathbb{S}) = \frac{1}{T}\sum_{t=2}^{T}$ equal$(S^t, F(S^{t-1}))$, where equal$(S, \hat{S})$ = 1 if for all output attributes k, s_k = round$(\hat{s}_k)$, and 0 otherwise. This measure can also serve to evaluate the quality of a source model F^{SRC} on target data $\mathbb{S}^{\text{TRG}}$, referred to in this thesis as *transfer model fitness* (TMF) of SRC to TRG, = accuracy$(F^{\text{SRC}}, \mathbb{S}^{\text{TRG}})$. These accuracy measures range from 0 to 1, with higher values denoting better models. To calculate a source model's TMF on target data, the target data input must be fed into the source model, and the model output must be compared against the target data output. Some object variables may appear in one game but not another. When a variable appears in the target but not the source, the basic assumption is that the source model ignores that variable by assuming a coefficient of zero. When a variable appears in the source but not the target, this variable in the source model assumes a value of zero when evaluating target data. In future work, these assumptions may be replaced by additional intelligence for variable mapping, discussed later in this section.

The TMF measure serves to estimate which source models are likely to transfer well to target object classes. In the example above, the agent notices that the Escape avatar behaves similarly to the Chase avatar – TMF is the measure of that behavioral similarity. A source selection algorithm might additionally use other measures such as the visual similarity of the icon used to represent the object. It could even use measures unrelated to how well the model fits to target data, such as the quantity of training in the source tasks or the frequency at which the model transfers successfully to other source object classes. Such additional measures could further improve performance on source selection but are left to future research.

3.6.3 Target Model as Ensemble of Source Models

The algorithm for transferring object class models is described in detail in this subsection. The algorithm starts with several trained object class models from source games. Then it observes some frames in a new target game. Some of the source models should predict later observations of the target game objects more accurately than new models trained from scratch on the observations made so far. Specifically, the assumption is that source models with high TMF to the target data should be more useful than those with low TMF. Therefore, for each object class in the target game, the algorithm builds a *transfer ensemble* out of both the pool of source models and the new scratch target model. The basic ensemble used is a forward model that makes predictions based on the weighted sum of its constituent forward models' predictions. Each of its constituents is assigned a weight as follows:

1. The scratch target model gets a nominal weight of 1.
2. Each source model j gets a nominal weight equal to $\mu\frac{b_j-\theta a}{M}$, where b_j is the source model's TMF, a is the scratch model's accuracy, μ and θ are tuning parameters, and M is the number of source models.
3. All models with non-positive weights are dropped from the ensemble.
4. The final weights are normalized by dividing the nominal weights by their sum.

The tuning parameters θ and μ control the skew given to the weights. The parameter θ takes a value between 0 and 1 and represents how much of the scratch model's accuracy is subtracted from the source models' scores. A higher θ eliminates the source models performing below the threshold and boosts the relative weights of the stronger source models compared to weaker ones. The parameter μ is a more general multiplier representing how much weight to allocate to the total of the source models relative to the scratch model. The values of these parameters are chosen based on what works well in previous experimentation. In this thesis, 30 games were withheld from final experiments and used for finding

good parameter values, setting θ and μ to 0.5 and 10, respectively.

The transfer ensemble is expected to predict target objects in out-of-sample data better than the scratch target model alone because of the inductive bias — the ensemble is biased toward models that work in other games.

3.6.4 Retraining the Source Models with Target Data

After selecting the source models and their relative weights in the transfer ensemble, *retraining* is a way to further specialize them to the target task and reduce negative transfer. Retraining entails adjusting the coefficients of the models through the same gradient descent method used to train them the first time, so that their predicted outputs better match the observed target data. Retraining fixes the mistakes made by the source models on target data while leaving intact the parts of the models uninformed by target data. Object attributes not encountered in the target training data will continue to influence the source model the same way as before retraining. To see why, recall that gradient descent updates model weights in proportion to $(\hat{s}_t^j - s_t^j)s_{t-1}^i$. Any input variable s_{t-1}^i not encountered in the target data will take a value of zero and not have its coefficients altered by gradient descent during retraining.

For example, consider the Chase model used to play Escape as discussed above. In Escape, there are hole objects that can kill an avatar that walks into them, but no such object exists in Chase. If a hole is encountered in Escape, the Chase model can be retrained to learn that the hole kills the avatar. Because the model coefficients are adjusted from their previous values rather than being reinitialized, the Chase model still correctly predicts that the avatar is blocked by walls without needing to encounter walls in Escape.

A transfer ensemble's source constituents are retrained only after their weights in the ensemble are determined. Their original weights relate to how likely the observed target object is to be of the same class as the source model, and setting the weights after retraining would cause them to be closer to each other than otherwise, which would dilute the relative

strength of models more likely to best predict the target object.

The experiments in this thesis show that retraining improves the accuracy of the transfer ensembles and reduces the number of cases and severity of negative transfer.

3.6.5 Variable Mapping for Relational Attributes

The problem of inter-task variable mapping is common in transfer learning problems where the variables in the source task do not correspond neatly to the variables in the target task. The object-oriented representation is chosen in this thesis because most objects use the same pool of attributes, averting the need to learn a mapping between variables in different objects. For example, internal state attributes such as directional movement, orientation, death, and player actions can be assumed to map directly from one object class to another.

However, every attribute that denotes a relation between objects relies on an indicator of object class that does not represent the same object class across all games. For example, the attribute $h^{N,2}$ represents adjacency with an object of class "2" on the Northern border, but object class "2" in one game may have nothing to do with object class "2" in another game. Future research in learning mappings between these indicators in relational attributes could help transfer work more often. Rather than pointing to specific object classes, relational attributes might point to learned families of object classes. For example, instead of attribute $h^{N,2}$, $h^{N,o_{\text{danger}}}$ might be used where o_{danger} is a label estimated from the behavior and look of the other object. For now, the cases where object classes indicated in relational attributes are mismatched are not enough to undermine the experiments in this thesis.

3.7 From Models To Actions

A forward model can be evaluated by its predictive accuracy. In this thesis, the metric used is the percentage of predictions the model gets exactly right. However, the ultimate goal in reinforcement learning problems such as video games is not just to make accurate

predictions, but to collect rewards. Rewards can include a large single payoff upon winning the game, a large single cost upon losing, or various changes to the final score accumulated throughout the game by killing enemies, grabbing coins, losing health, and so on.

An agent can use a planner on top of a model to help it collect rewards. The planner uses the model to simulate what future states might look like given the considered actions. The model predicts the next state from the current state and a given action. To plan one step ahead, the planner considers what the predicted next state will be for all possible actions and chooses the action that results in the best future state. The agent therefore needs a way to evaluate the transitions to future states. One way is to manually specify a hand-written reward function. Another possibility is to learn a termination model that predicts the game outcome from state variables. A more comprehensive method is to learn a value table or value function so that states other than those immediately leading to termination can be evaluated. An agent can use any of these evaluation methods in conjunction with the forward model and planner in order to guide its selection of actions while playing.

One important question is whether more accurate models are actually more useful for selecting good actions. It is not always the case that lower prediction errors result in higher-performing policies [23]. For example, a model that overestimates the risk of death from some cause may act more conservatively and be more likely to survive than a model whose estimates are a closer fit to the observed data. Because the best assessment of the presented approach is in the context of the whole game-playing pipeline, a simple planner and manually designed reward function are integrated for some of the experiments and described in detail in those sections.

Chapter 4

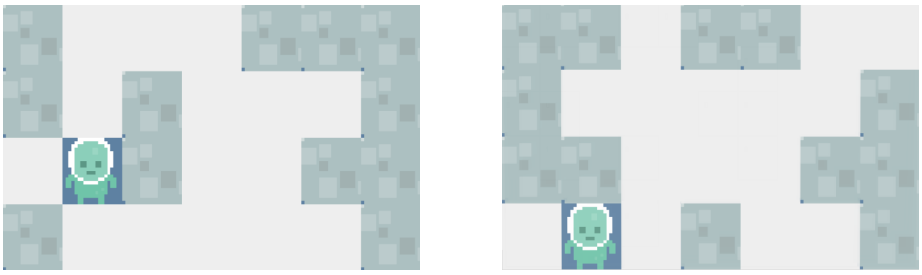
Experiments and Results

Out of the 60 GVG-AI games, 30 were used for exploratory testing and tuning of the system, and the other 30 withheld for experiments. The reason for this division was to prevent biasing the results of the experiments. The first set was used to develop the approach described in the previous chapter. The second set is to test that this approach works on an out-of-sample segment of the domain as well. All experiments described in this chapter were performed on this second set.

4.1 Grid-Oriented Versus Object-Oriented Modeling

The first experiment is to confirm that the object-oriented approach does in fact produce models that generalize better to new experiences. The hypothesis is that an agent that learns a model predicting future states of a game from its grid activations will fail to generalize to true out-of-sample observations, while an agent that predicts future states using learned object models will succeed in generalizing.

This experiment uses custom levels A and B of the game Labyrinth, pictured in Figure 4.1. These two levels separate in-sample data used for training and out-of-sample testing data. The experiment consists of two agents – a grid-oriented model learner and an



(a) Labyrinth in-sample level A (b) Labyrinth out-of-sample level B

Figure 4.1: Different levels of the same game with the same game rules are used for training and testing. Using these different levels ensures that the game states used for training do not overlap with game states used in testing.

object-oriented model learner – going through three phases:

1. Train a forward model for level A using 500 frames of data with random action selection until perfect accuracy on training data is achieved;

2. Test the forward model on 500 new frames of level A using random action selection and record accuracy; and

3. Test the forward model on 500 frames of level B using random action selection and record accuracy; also record the prediction accuracy just for the location of the avatar.

The testing accuracy recorded in step 2 is on data that contains many of the same states as the training data, comparably to what is often reported as testing accuracy in existing research on general video game playing. The ability to test on a different level as in Step 3 allows for an assessment of generalization ability on out-of-sample data.

The object-oriented model learner is the same as described in the approach of this thesis. The grid-oriented model learner is a neural network pictured in Figure 4.2 and constructed as follows:

- An input layer of 62 nodes represents the player action and instances of each object class at each location in the game;

- A hidden layer of 100 sigmoidal nodes takes as inputs the weighted sums of the values of the input layer; and
- An output layer of 56 sigmoidal nodes takes as inputs the weighted sums of the outputs of the hidden layer. Rounded values of the output nodes represent the existence of instances of each object class at each location in the next frame. This neural network additionally uses bias nodes and is trained with back-propagation to minimize cross entropy.

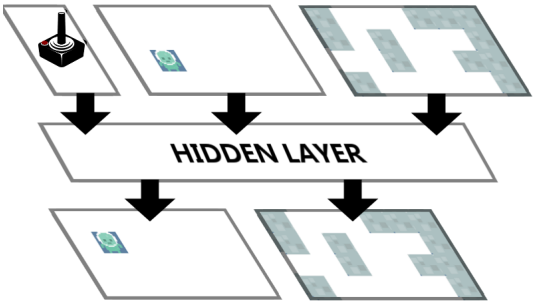


Figure 4.2: The grid-oriented model uses variables representing each object class at each location. Taking the form of a neural network with one hidden layer, the model predicts the next state of the game from the previous state and player action. The total number of weights is polynomial in the size of the grid times number of object classes because the size of the hidden layer must scale with the size of the input in order for the model to predict accurately.

Table 4.1: Prediction accuracy achieved in in-sample training, in-sample testing, and out-of-sample testing for grid-oriented and object-oriented model learners. The grid-oriented model does not generalize to the out-of-sample level, while the object-oriented model does.

	Training accuracy	Testing accuracy		Avatar only
Agent	Level A	Level A	Level B	Level B
Grid-oriented	100%	100%	0%	0%
Object-oriented	100%	100%	100%	100%

As seen in Table 4.1, the grid-oriented neural network forward model trained on level A achieves perfect accuracy on fresh testing data from level A. However, when using

the same model to predict states in level B, accuracy drops to zero percent. Even the movement of the avatar by itself becomes unpredictable. In contrast, the object-oriented model is able to generalize perfectly to the new level. To gain a better understanding of why the object-oriented representation learns more general rules than the grid-oriented representation, consider an example:

A grid-oriented agent learns that pressing RIGHT when the avatar is at (0,0) changes the state of the game so that the avatar is at (1,0). From this observation alone it cannot predict that pressing RIGHT again from (1,0) will move the avatar to (2,0) because it has never had the exact experience of moving RIGHT from (1,0) before. An object-oriented agent learns that pressing RIGHT moves the avatar object to the right. It expects this rule to hold even when the avatar is in new locations.

The results of this experiment demonstrate the importance of using fully out-of-sample data to test generalization ability and suggest that object-oriented representations are preferable to grid-oriented representations.

4.2 Testing Transfer Ensembles of Object Class Models

The purpose of the experiments in this section is to test the generalization ability of transfer ensembles compared to models learned from scratch. The hypothesis is that, after observing some in-sample training data from target games, transfer ensembles achieve higher accuracy on out-of-sample target data than scratch models. Particularly, this outperformance should be more pronounced when the in-sample data is sparse or noisy. Furthermore, the use of a retraining phase after the construction of the ensembles should help improve performance and reduce negative transfer. In the experiments, sparse training conditions are produced by limiting the observed in-sample training frames to a low number T. Noise, such as real-life sensor malfunction, is simulated by randomly toggling the values of the directional movement attributes observed for each object at each in-sample frame according to probability ϵ. Each experiment produces 100 subsequent frames for each target game using random player

actions and measures out-of-sample accuracy for each object class model, comparing three approaches: training from scratch, constructing a transfer ensemble without retraining, and constructing a transfer ensemble with retraining. The number of source games used for each transfer ensemble is $N = 6$ in each experiment. Additionally, in order to gauge sensitivity to the choice of source games, each experimental setup is tested three times using a different set of six source games selected at random without replacement. In summary, the following are the variables of interest that are compared across experiments:

Model Type: Scratch, Transfer, Transfer+Retraining;

Training Frames T: 10, 100;

Noise ϵ: 0.0, 0.05;

Number of Sources N: 6; and

Set of Source Games: {Aliens, Bait, Boloadventures, CamelRace, Catapults, Chase}, {Defender, Eggomania, Missilecommand, Jaws, Hungrybirds, Labyrinth}, and {Brainman, Butterflies, Cakybaky, Chipschallenge, Escape, Crossfire};

Each experiment tests the three model types on a given combination of values of the other parameters, for a total of three times 12 experiments, using the following procedure:

1. Separate the 30 games into N source games and $(30 - N)$ target games;
2. Train and store source object class models from scratch using 500 frames of the source games; and
3. For each object class in each target game, up to 150 total object classes:
   (a) Collect target training data $\mathbb{S}$ by observing the first T frames of the game with sensor error ϵ;
   (b) Train a scratch model on $\mathbb{S}$;

(c) Calculate TMF of each source model with $\mathbb{S}$;

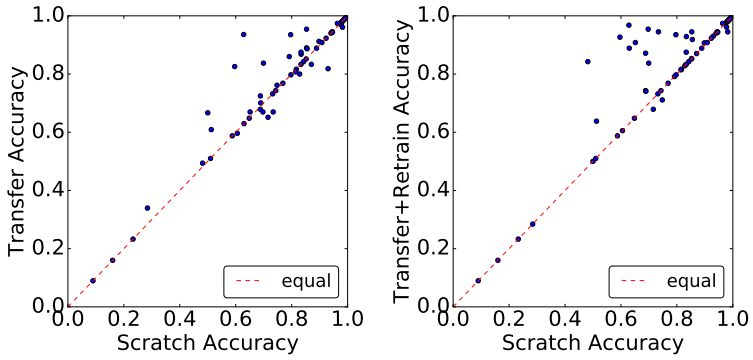
(d) Build a transfer ensemble model from the trained scratch model and the existing source models, using TMF to inform weights;

(e) Copy the transfer ensemble model and retrain the new copy using the target training data.

(f) Collect target test data $\mathbb{S}'$ by observing 100 frames of the game, using a different level from the one used for training; and

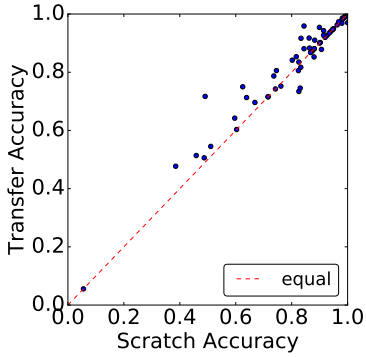
(g) Record prediction accuracy of the scratch target model, the transfer ensemble model, and the retrained transfer ensemble model on test data $\mathbb{S}'$.

The main measure of success is the outperformance in forward prediction accuracy of transfer models over scratch models in out-of-sample frames for each object class in each target game. To reject the null hypothesis of the differences being due to chance, a t-statistic is used to compute a one-sided p-value that must be less than 0.05.

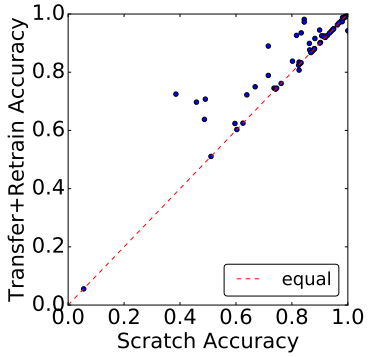


(a) $T = 10$, $\epsilon = 0\%$, S_1

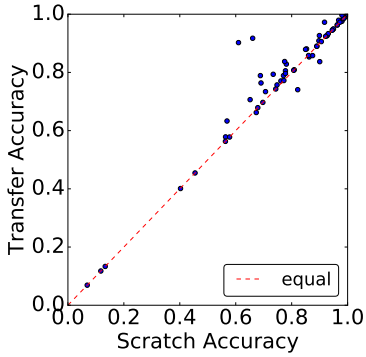
(b) $T = 10$, $\epsilon = 0\%$, S_1



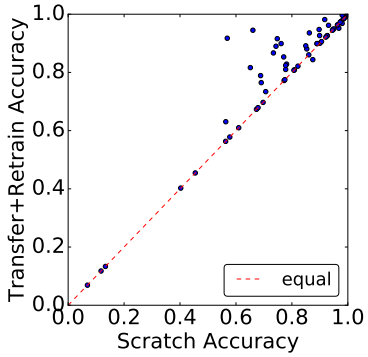
(c) $T = 10$, $\epsilon = 0\%$, S_2



(d) $T = 10$, $\epsilon = 0\%$, S_2

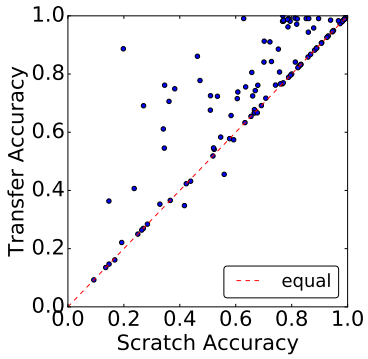


(e) $T = 10$, $\epsilon = 0\%$, S_3

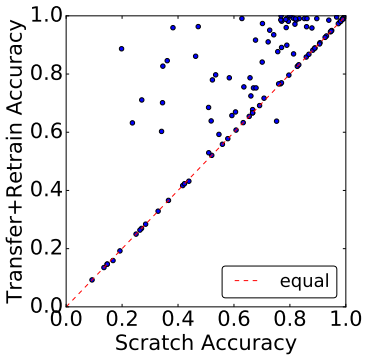


(f) $T = 10$, $\epsilon = 0\%$, S_3

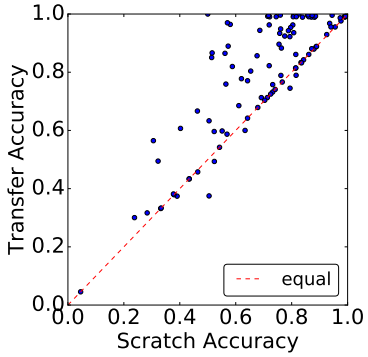
Figure 4.3: Test accuracy for scratch models versus transfer models, without retraining on the left and with retraining on the right. Points represent object classes, which come from all of the target games. Points appearing on the line or in the top-left half of the plot indicate transfer does as well or better than scratch. Transfer outperforms scratch for many object classes and reduces accuracy very rarely.



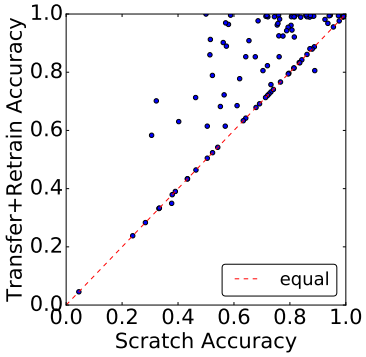
(g) $T = 10$, $\epsilon = 5\%$, S_1



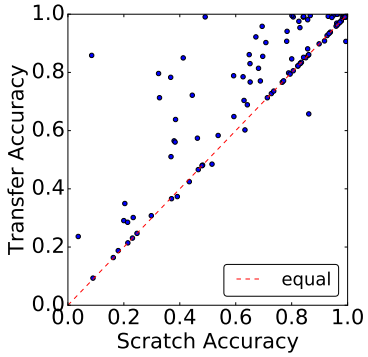
(h) $T = 10$, $\epsilon = 5\%$, S_1



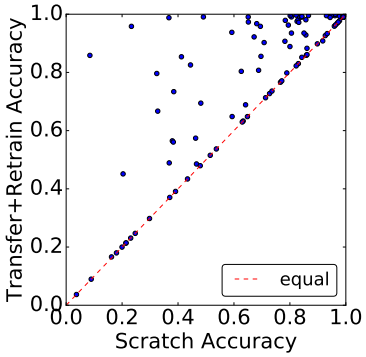
(i) $T = 10$, $\epsilon = 5\%$, S_2



(j) $T = 10$, $\epsilon = 5\%$, S_2

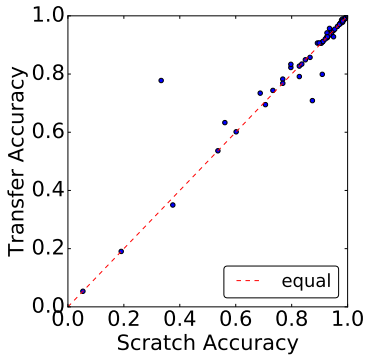


(k) $T = 10$, $\epsilon = 5\%$, S_3

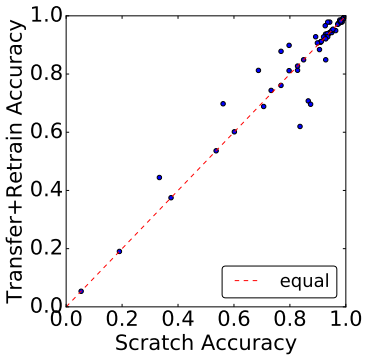


(l) $T = 10$, $\epsilon = 5\%$, S_3

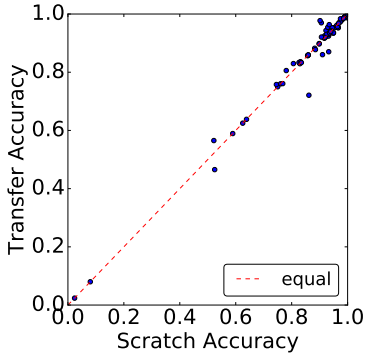
Figure 4.3: *Continued.*



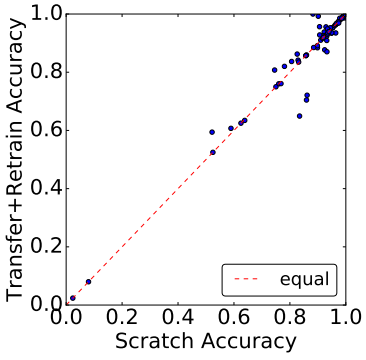
(m) $T = 100$, $\epsilon = 0\%$, S_1



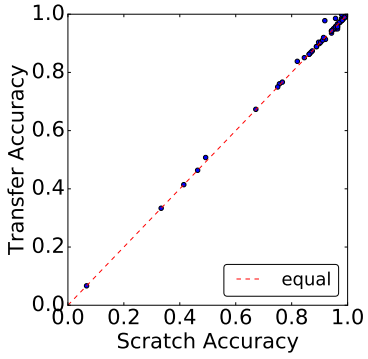
(n) $T = 100$, $\epsilon = 0\%$, S_1



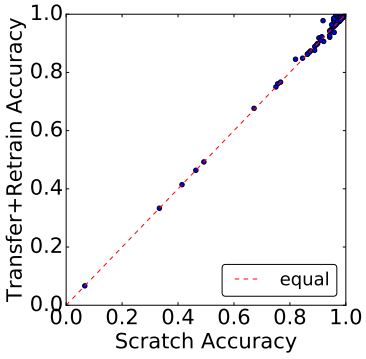
(o) $T = 100$, $\epsilon = 0\%$, S_2



(p) $T = 100$, $\epsilon = 0\%$, S_2

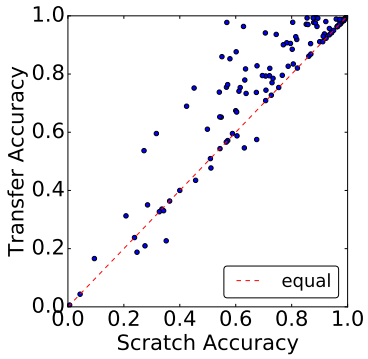


(q) $T = 100$, $\epsilon = 0\%$, S_3

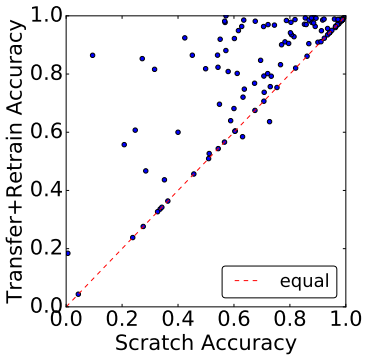


(r) $T = 100$, $\epsilon = 0\%$, S_3

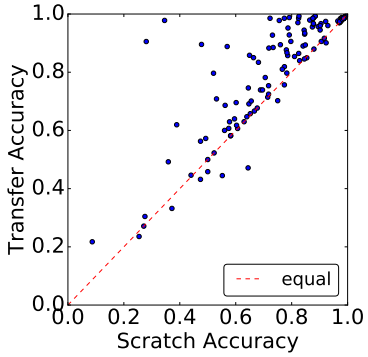
Figure 4.3: *Continued.*



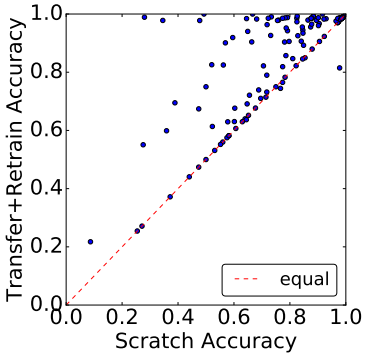
(s) $T = 100$, $\epsilon = 5\%$, S_1



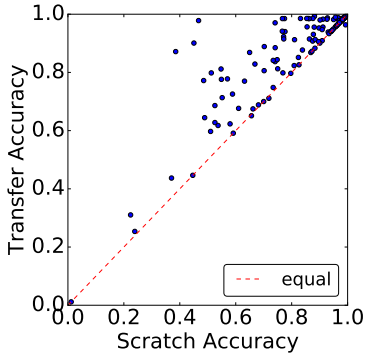
(t) $T = 100$, $\epsilon = 5\%$, S_1



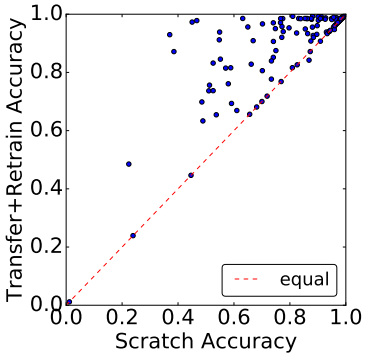
(u) $T = 100$, $\epsilon = 5\%$, S_2



(v) $T = 100$, $\epsilon = 5\%$, S_2



(w) $T = 100$, $\epsilon = 5\%$, S_3



(x) $T = 100$, $\epsilon = 5\%$, S_3

Figure 4.3: *Continued.*

Table 4.2 shows that the average increase in accuracy is statistically significant for the adverse training scenarios. Based on the rejection of the null hypothesis that the im-

Table 4.2: Accuracy of tested models for each experimental setup with training frames T, simulated noise ϵ, and source set S. Results reported include scratch mean μ_s, transfer mean μ_t, transfer+retraining mean μ_{tr}, scratch avatar mean μ_s^a, transfer avatar mean μ_t^a, transfer+retraining avatar mean μ_{tr}^a. Metrics of interest include t-statistics for differences in object class model accuracy between transfer and scratch t_t and between transfer+retraining and scratch t_{tr}. Bold p-values (p_t and p_{tr}) indicate rejection of the one-tailed null hypothesis that the improvement is due to chance. Improvement in accuracy from using transfer is statistically significant, particularly when training data is sparse or noisy.

T	ϵ	S	μ_s	μ_t	μ_{tr}	μ_s^a	μ_t^a	μ_{tr}^a	t_t	t_{tr}	p_t	p_{tr}
10	0%	1	0.90	0.91	0.92	0.74	0.74	0.80	2.28	3.53	**1%**	**<1%**
10	0%	2	0.92	0.93	0.94	0.75	0.78	0.82	2.73	3.46	**<1%**	**<1%**
10	0%	3	0.91	0.92	0.92	0.78	0.80	0.85	2.55	3.70	**<1%**	**<1%**
10	5%	1	0.76	0.82	0.83	0.65	0.71	0.74	6.52	6.89	**<1%**	**<1%**
10	5%	2	0.79	0.86	0.87	0.61	0.69	0.71	7.59	7.96	**<1%**	**<1%**
10	5%	3	0.78	0.83	0.85	0.66	0.72	0.73	5.81	6.54	**<1%**	**<1%**
100	0%	1	0.94	0.94	0.94	0.89	0.90	0.91	0.68	0.12	25%	45%
100	0%	2	0.94	0.94	0.94	0.75	0.73	0.67	-0.2	-0.3	42%	39%
100	0%	3	0.95	0.95	0.95	0.91	0.91	0.92	1.95	2.50	**3%**	**<1%**
100	5%	1	0.77	0.82	0.86	0.65	0.75	0.81	6.87	7.63	**<1%**	**<1%**
100	5%	2	0.79	0.85	0.87	0.65	0.72	0.78	6.66	7.62	**<1%**	**<1%**
100	5%	3	0.84	0.89	0.92	0.72	0.78	0.82	7.40	8.16	**<1%**	**<1%**

provements from transfer are due to chance, it can be concluded that the transfer ensemble approach for learning models of object classes is sound. Figure 4.3 displays how the transfer ensemble models compare in out-of-sample accuracy against models trained from scratch. Each dot represents one object class model; scratch and transfer perform equally when a dot falls on the line, transfer outperforms when a dot is in the upper-left, and scratch outperforms when a dot is in the lower-right. These graphs show that improvement due to transfer is consistent across many games with rare occurrences of negative transfer.

As shown in Table 4.2, when the amount of training data is very small but clean, the average difference between scratch and transfer performance is only about two percent. However, the average difference understates the significance of the improvement using transfer. Many object classes are easy enough to model so that scratch achieves perfect accuracy. For example, wall objects never move and are modeled with perfect accuracy by

scratch in all the tested target games. More important is the improvement in object classes that are hard to model, such as the avatar, which behaves with varying complexity depending on the game. When the amount of training data is very small but the data is clean, transfer achieves higher accuracy of about seven percent for the avatar models. The results also suggest that transfer can be very helpful when dealing with noisy data. Overall, these results strongly support the hypothesis that using transfer ensembles with retraining lead to improved out-of-sample accuracy for many object classes, with very little negative transfer.

4.3 Exploration

The exploration experiments test how well scratch and transfer models perform relative to each other and relative to a random action-taking agent on the task of exploring the environment. Agents perform this task on three levels of the game Labyrinth. This game is well-suited for these experiments because the agent must guide the avatar to reach a destination through maze-like levels containing a few fatal spike tiles.

The purpose of these experiments is similar to that of the exploration experiments by Oh et al. [27], who showed how an agent that learns a forward model to predict one step ahead is able to explore the environment more efficiently than a random agent in the game of Ms. Pacman. The main difference is that in these experiments, the model-learning agents are only given a very small number of frames of training experience before the evaluation phase begins.

First, agents are given either 10, 50, or 100 frames of training before being evaluated on a fresh 500 frames. If the avatar dies or reaches the goal at any time, the game is restarted with the avatar in its original position so that the number of frames used is unaffected. In all setups, the transfer agent is built using an ensemble of six random source games other than Labyrinth, with 500 frames of training for each source, and retrained on the 10/50/100 frames of target data. Termination models are trained in addition to object class models for both scratch and transfer in source and target games in order to help predict death.

After training, agents go through a testing phase of another 500 frames. During this phase, agents use their forward models to choose one-step actions most likely to take them to novel or least-recently visited states. Their decision-making works as follows. The agent remembers each unique game state it visits and the time frame at which it was last visited. For each action the agent predicts the next game state by using its object class models to predict the next state of each game object. Model outputs are treated as probabilities of setting the corresponding object variables to one. Treating forward predictions as probabilistic samples in this way helps agents avoid getting stuck. The value of each action considered at each frame by an agent is calculated as $1 - \frac{t_{\hat{S}}}{t_A}$, where t_A is the current time frame of the game and $t_{\hat{S}}$ is the last visited time frame of the predicted next state ($t_{\hat{S}} = 0$ if the state has never been visited). If the agent predicts death the value is -1. At each time frame the agent chooses whichever action has the maximum value. At the end of the testing phase, the total number of unique states visited are counted and used as the metric of evaluation.

After the testing phase, an additional 500-frame phase is run to measure prediction accuracy as in the previous experiments. During this phase, all agents take random actions rather than informed ones, in order to ensure fair comparison. The purpose is to determine the relationship between model accuracy and actual performance on an important task requiring action selection. The improvements in model accuracy found in the previous experiments can therefore be related to meaningful improvements in agent performance. For each of the three levels of Labyrinth and each of the three training setups, the experiment is run five times, each time collecting the accuracy of the scratch model, the accuracy of the transfer model, the explored range of the random agent, the explored range of the scratch agent, and the explored range of the transfer agent.

Table 4.3: Results of the exploration experiments. The Training column denotes the number of game frames observed before evaluation. Accuracy$_S$ and Accuracy$_T$ are the out-of-sample scratch and transfer model accuracy measurements, respectively. The Explored$_R$, Explored$_S$, and Explored$_T$ columns report the number of unique states explored during testing by agents using random action selection, the scratch model, and the transfer model, respectively. This data again confirms that transfer models achieve higher accuracy than scratch models when there is little training data, and it further shows that the more accurate models can be used for more efficient exploration.

Map	Training	Accuracy$_S$	Accuracy$_T$	Explored$_R$	Explored$_S$	Explored$_T$
Labyrinth0	10	0.6620	0.9980	25	8	75
Labyrinth0	10	0.7040	0.9980	8	9	57
Labyrinth0	10	0.672	0.7385	21	32	60
Labyrinth0	10	0.8124	0.8124	6	18	67
Labyrinth0	10	0.6940	0.7485	30	7	60
Labyrinth1	10	0.7186	1.0000	17	24	64
Labyrinth1	10	0.7880	0.8140	20	18	22
Labyrinth1	10	0.8240	0.8323	20	21	24
Labyrinth1	10	0.6647	0.9220	12	23	44
Labyrinth1	10	0.8403	0.6407	12	17	49
Labyrinth2	10	0.6460	0.8500	29	21	39
Labyrinth2	10	0.7505	1.0000	16	38	37
Labyrinth2	10	0.6966	1.0000	18	17	35
Labyrinth2	10	0.6906	1.0000	17	12	40
Labyrinth2	10	0.6580	0.7720	20	24	12
Labyrinth0	50	0.8024	0.9561	13	14	54
Labyrinth0	50	0.8220	0.7146	18	6	41
Labyrinth0	50	0.8640	0.9960	29	76	60
Labyrinth0	50	0.9281	0.9980	24	37	55
Labyrinth0	50	0.8084	1.0000	16	29	29
Labyrinth1	50	0.8543	0.8902	33	17	61
Labyrinth1	50	0.8543	1.0000	20	19	70

Continued on next page

Table 4.3 – *Continued from previous page*

Map	Training	Accuracy$_S$	Accuracy$_T$	Explored$_R$	Explored$_S$	Explored$_T$
Labyrinth1	50	0.9020	1.0000	20	18	43
Labyrinth1	50	0.8762	1.0000	20	15	40
Labyrinth1	50	0.9641	0.6407	10	16	53
Labyrinth2	50	0.7240	0.9980	35	12	38
Labyrinth2	50	0.8500	0.9581	32	15	45
Labyrinth2	50	0.9640	1.0000	32	41	41
Labyrinth2	50	0.9480	0.9058	17	39	45
Labyrinth2	50	0.9000	0.8320	27	53	41
Labyrinth0	100	0.9980	1.0000	20	84	89
Labyrinth0	100	0.8084	0.7884	32	13	31
Labyrinth0	100	1.0000	0.8583	6	71	56
Labyrinth0	100	0.9681	0.7804	39	79	61
Labyrinth0	100	1.0000	1.0000	9	68	65
Labyrinth1	100	0.8583	0.9820	23	15	44
Labyrinth1	100	0.8560	1.0000	31	25	39
Labyrinth1	100	0.8623	1.0000	35	18	69
Labyrinth1	100	0.9960	1.0000	19	66	58
Labyrinth1	100	0.9920	0.9641	20	65	51
Labyrinth2	100	0.9521	0.8563	27	47	38
Labyrinth2	100	1.0000	0.9601	17	43	41
Labyrinth2	100	0.8084	0.9481	20	19	43
Labyrinth2	100	0.8838	0.9980	21	12	41
Labyrinth2	100	0.9960	1.0000	21	38	71

Table 4.4 and Figure 4.5 show how scratch and transfer agents perform in exploring three levels of Labyrinth given 10, 50, and 100 initial frames of training. The agents trained from scratch on only ten frames of the game are not highly accurate in out-of-sample experience and struggle to perform better than random exploration. In contrast, the transfer

Table 4.4: Improvement in accuracy and exploration over random actions, averaged over level and training quantity, for the scratch and transfer approaches. It is clear from the averages that transfer outperforms scratch in exploration even when they are tied for accuracy, as in the results of the 100-frame training scenarios. The conclusion is that transfer leads to better accuracy and exploration performance.

Map	Training	Accuracy$_S$	Accuracy$_T$	Explored$_{S-R}$	Explored$_{T-R}$
Labyrinth0	10	0.71	0.86	-3.2	45.8
Labyrinth1	10	0.77	0.84	4.4	24.4
Labyrinth2	10	0.69	0.92	2.4	12.6
Labyrinth0	50	0.84	0.93	12.4	27.8
Labyrinth1	50	0.89	0.91	-3.6	32.8
Labyrinth2	50	0.88	0.94	3.4	13.4
Labyrinth0	100	0.95	0.86	41.8	39.2
Labyrinth1	100	0.91	0.9	12.2	26.6
Labyrinth2	100	0.93	0.95	10.6	25.6

agents are more accurate, supporting the results of the previous experiments, and are also able to explore much more efficiently.

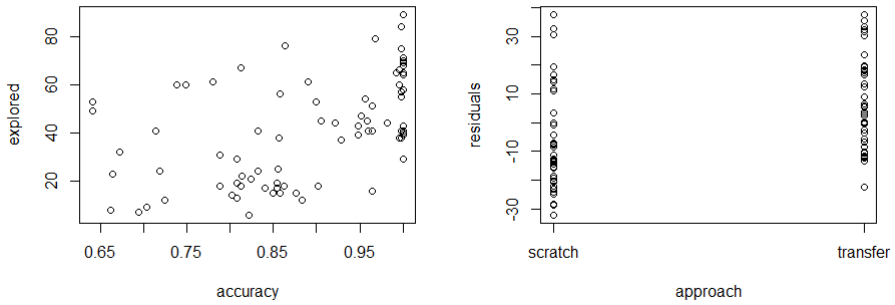
To understand the relationship between model prediction accuracy and exploration efficiency, the explored range of each model-based agent of each run is compared with the measured accuracy of that agent in that run, as shown in Figure 4.4a. Explored range is regressed on accuracy as $y_{\text{explore}} \sim \alpha + \beta x_{\text{accuracy}} + \epsilon$, thus testing the hypothesis that exploration performance is predictable from prediction accuracy.

The most likely estimates of the intercept term α and the coefficient β are -43.11 and 94.12, respectively, both with p-values under 1%. The standard error of the residual ϵ is 17.84. This result supports the hypothesis, with each percentage increase in accuracy resulting in roughly one additional explored state. However, there is substantial remaining error left unexplained by accuracy.

Table 4.4 shows that as the number of training frames increases to 100, scratch models catch up in accuracy to transfer models. This effect is consistent with the previous set of experiments with 100 frames of training and no noise. Interestingly, however, the transfer agents still explore more efficiently on average than the scratch agents, despite not

being any more accurate. Grouping the residuals from the above regression by the approach used, shown in Figure 4.4b, reveals that the scratch residuals have a mean of -5.88 and standard deviation of 16.97 while the transfer residuals have a mean of 5.48 and standard deviation of 16.70. This difference in means is statistically significant with a two-sample t-value of 3.22 and p-value of under 1%. Thus the difference between the explored range achieved in each run and the explored range that would be expected given the measured accuracy is higher in expectation by over eleven states when the agent uses a transfer model compared to when it uses a scratch model. Eleven states is a substantial advantage in this context, considering the average number of states explored by a random agent across these experiments is only about 21. Not only does transfer yield a better accuracy than scratch when the training size is small, but it results in additionally better performance in the exploration task unexplained by the better accuracy.

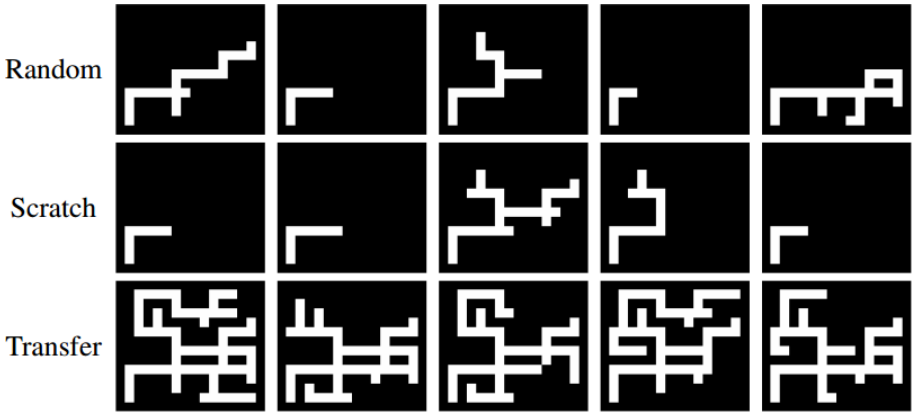
One possible explanation for the outperformance unexplained by accuracy is that the transfer agent may be more accurate in the more important predictions than the scratch agent. Labyrinth maps contain a few stationary spike tiles that kill the avatar on contact. If the avatar dies, the agent must restart from the original position. The transfer agent may more accurately predict this deadly interaction than a scratch agent when no spike traps appear during training because the transfer agent is composed of some models that predict death from contact with foreign objects. This advantage might have minimal impact on total prediction accuracy because the accuracy measure is diluted by other predicted behaviors. However, being able to avoid death allows the transfer agent to keep exploring while the scratch agent has to start over.



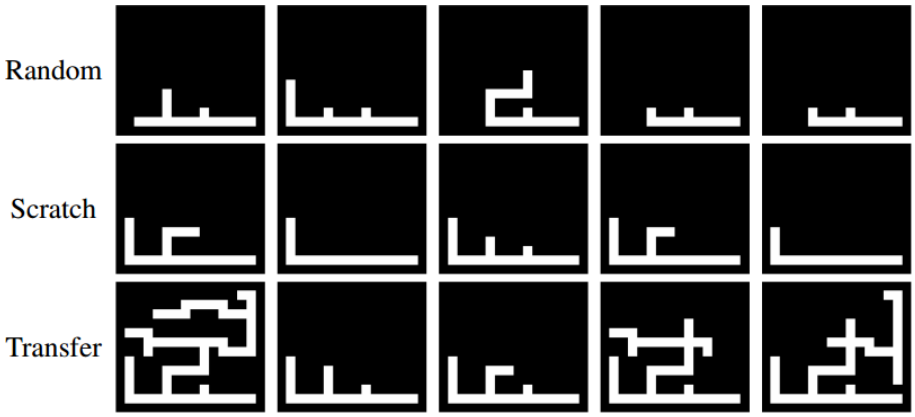
(a) Number of unique states explored versus model prediction accuracy

(b) Residuals of exploration regressed on accuracy, grouped by approach

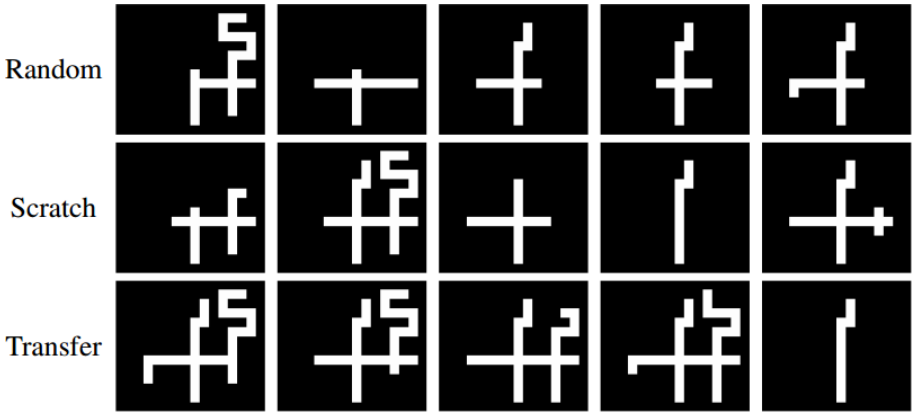
Figure 4.4: Scatter plots depicting relationships between explored range, measured accuracy, and approach type over all exploration experiments. Linear regression shows statistically significant positive relationship between model accuracy and exploration ability. Residual error is partially explained by approach type, with agents that use transfer models outperforming ones that use scratch models even after accounting for accuracy. This test confirms that transfer agents perform better than scratch overall, with some of that outperformance explained by improvement in accuracy.



(a) Labyrinth LVL0, 10 training frames

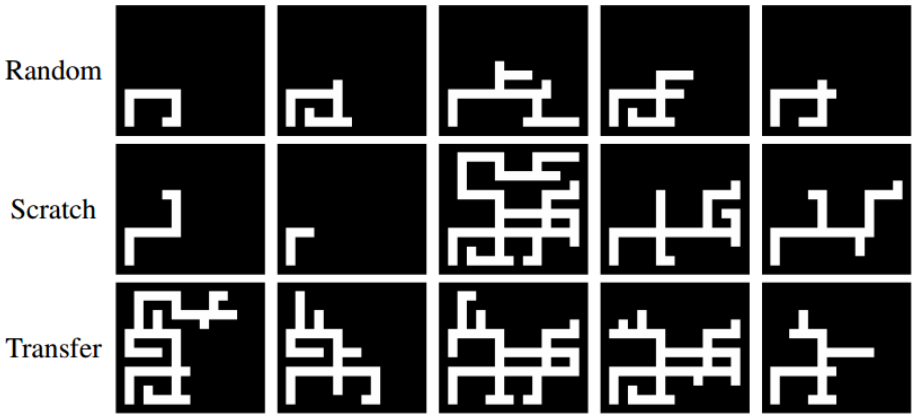


(b) Labyrinth LVL1, 10 training frames

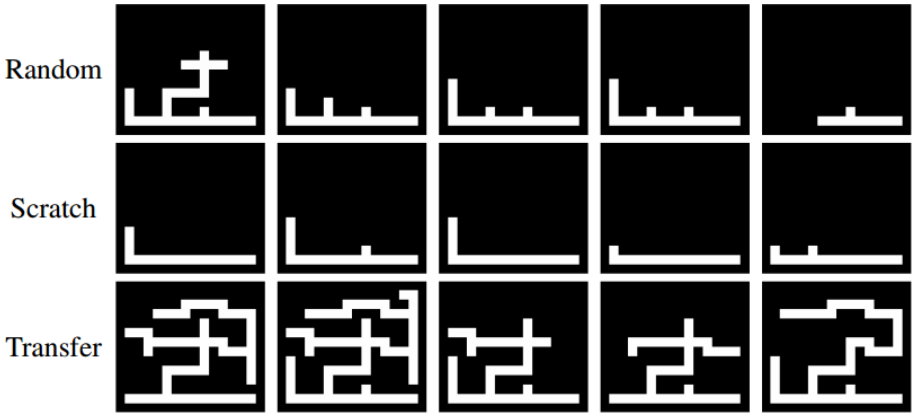


(c) Labyrinth LVL2, 10 training frames

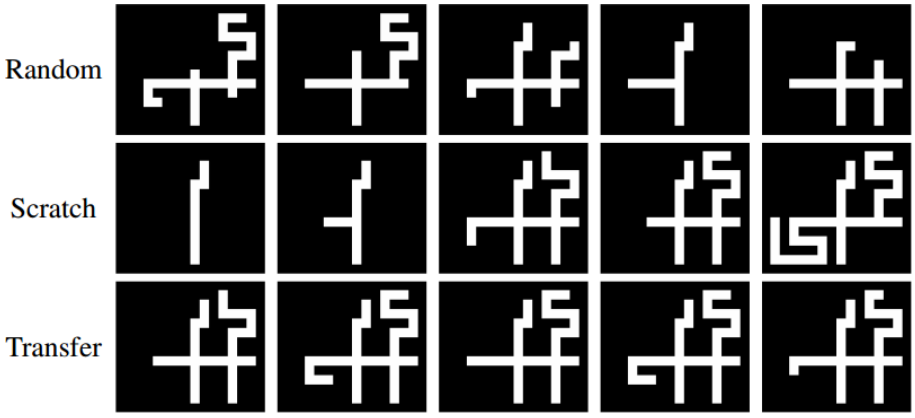
Figure 4.5: Trajectory maps of avatar during 500-frame test phase, five runs for each setup. The maps show that transfer agents usually explore the most space.



(d) Labyrinth LVL0, 50 training frames

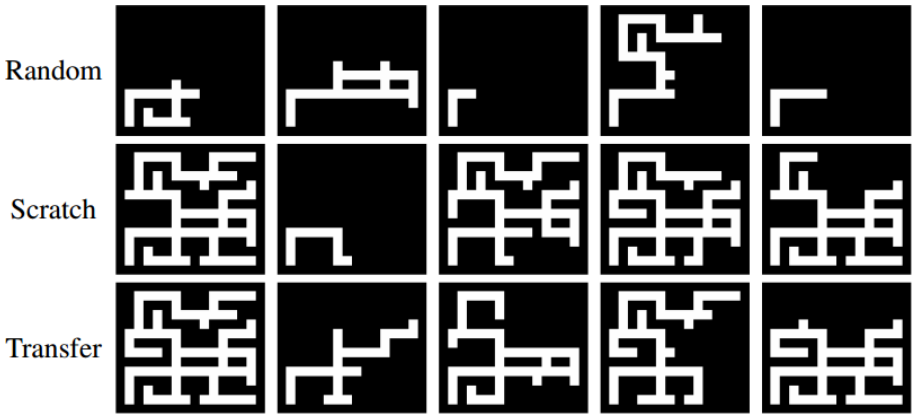


(e) Labyrinth LVL1, 50 training frames

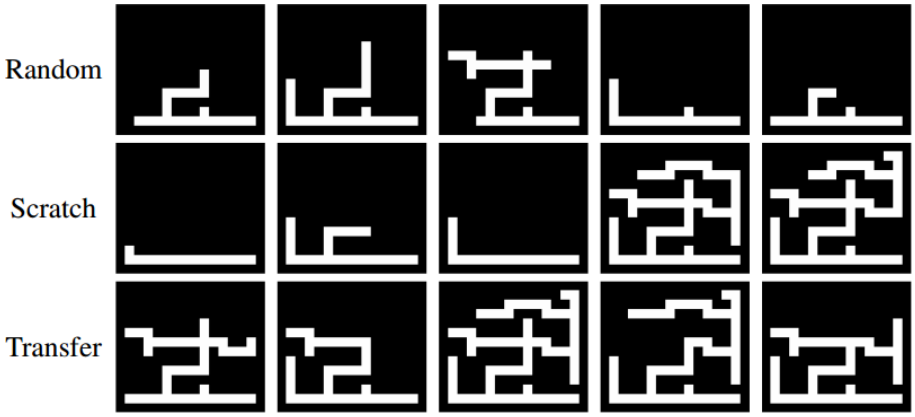


(f) Labyrinth LVL2, 50 training frames

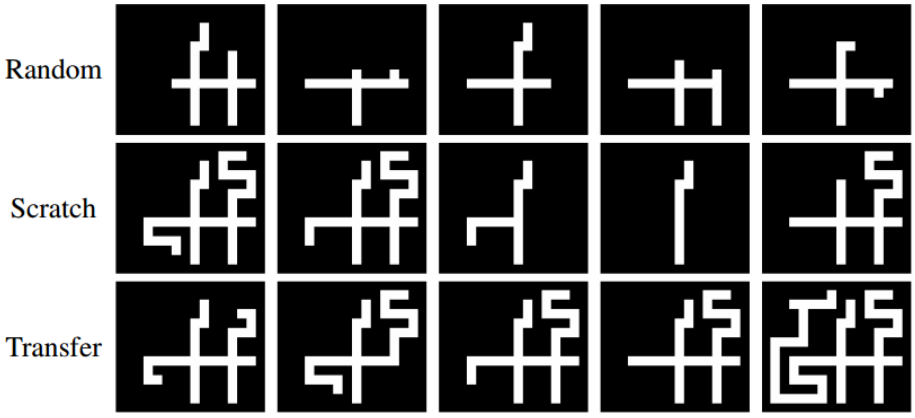
Figure 4.5: *Continued.*



(g) Labyrinth LVL0, 100 training frames



(h) Labyrinth LVL1, 100 training frames



(i) Labyrinth LVL2, 100 training frames

Figure 4.5: *Continued.*

Chapter 5

Discussion and Future Work

The first finding of this thesis is that object-oriented models of the video game domain generalize better than grid-oriented models. Even in a small, simple game of moving an avatar around a room with some walls, a grid-oriented agent that learns a perfect model of one level of this game fails to generalize to a different level where the walls are simply in different locations. On the other hand, the object-oriented agent is able to generalize to the new level. This agent learns how objects in the game move and interact with each other, which continues to be helpful when the states of the game are different to the states encountered during training.

On the other hand, one of the attractions of the grid-oriented representation is that it requires fewer assumptions and less prior knowledge. In order to produce an object-oriented representation, the designer must specify object class attributes and the method of translating between the game state and the object instance information. In this thesis, these design choices were made manually, but in future research it would be interesting to explore how the system could automatically determine when an object-oriented representation is appropriate, choose attributes, and translate between representations. In practice, some grid-based approaches boost generalization ability through constraints such as convolutional layers [26; 27], which force features learned in one region of the grid to extend

to the rest of the grid. It would be worth exploring how the methods and experiments in this thesis might apply to an approach that uses segments of convolutional neural networks rather than object classes.

The results of the second set of experiments showed how more accurate predictions can be made using object class models built as transfer ensembles compared to models learned from scratch. A retraining step further improves accuracy and reduces negative transfer. When the target data is clean and plentiful, scratch models are already accurate, and potential improvement from transfer is small. Rather, transfer is more useful when a scratch model is likely to generalize poorly because the data is limited or noisy. The noise in these experiments is an artificial simulation of the real errors seen when extracting objects in other domains such as Atari. One direction for future work is to compare against other methods in dealing with noisy data and partially observable state spaces.

The improvement from using transfer learning holds regardless of the set of source games used. Out of six randomly chosen games, it is likely that there are transferable object classes in at least one game. Some games may not contain any useful object classes, but including them in the source set does not undermine the performance because of the safeguards against negative transfer.

The final set of experiments tested the usefulness of object models learned from scratch and those learned using the transfer approach on the common task of exploring the environment. Exploration is often used in reinforcement learning problems, and maze-like environments such as the Labyrinth game used in these experiments are typical tests of exploration ability. Prior research has already shown how agents can use learned models of their environment to inform exploration. Models help predict which actions are most likely to lead to unexplored states, and even predictions of a single frame forward are helpful.

The results of the exploration experiments support the hypothesis that agents with more accurate models perform better. Transfer agents not only predict more accurately but also explore more efficiently than scratch agents. On top of the improved performance

explained by the improved accuracy, using transfer yields a significant improvement in exploration ability unexplained by the prediction accuracy alone. This extra performance boost may be due to transfer models having better accuracy in moments more critical to performance, for example when needing to avoid a threat to the agent's life.

This thesis addresses the question of using very few samples to learn to play video games, where samples are counted as observed frames from the game. An alternative way to consider this question is to count samples as whole episodes of a game. In this case, rather than worrying about the cost of collecting new observations, the agent must worry about the cost of additional lives. The two goals are related because having fewer lives should generally imply having fewer observations. If there is a limited number of lives available, it becomes much more important for a model-learning agent to avoid death in order to collect a higher total number of observations on which to train. Transferred knowledge could be particularly useful in this problem because it can help the agent guess possible causes of death. Future work should include experiments comparing how long scratch agents and transfer agents can survive in unfamiliar games.

There are several potential improvements to the presented methods. First of all, the attributes and form of the object models are not sufficient to capture all behaviors seen in GVG-AI games. Refinement of the object models should lead to better overall accuracy and more potential for successful transfer. Second, a method for variable mapping can help generate more viable candidate models for transfer. This problem may be approached by looking at some of the existing methods for *inter-task mappings* [35]. Third, there are several promising methods for further informing source selection, such as by looking at object visual similarity or by measuring how well object models transfer between source tasks.

Future work will lead to a full integration of these object model-learning and transfer-learning methods with other reinforcement learning methods, application to other sequential decision-making problems, and continued evaluation using sparse and noisy training

conditions and fully out-of-sample test data.

Chapter 6

Conclusion

This thesis investigated the problem of learning general models of video games from limited training data using a transfer-learning approach. The GVG-AI video game domain consists of a wide range of games with diverse challenges that test the generality of the approach. Crucially, its games also have stochastic behaviors and multiple levels, which test the generality of agents learned by the approach. This research makes the following contributions:

- Rigorous testing of general game agents on out-of-sample data guaranteed not to overlap with in-sample training data;
- Agents that learn useful models in the general video game domain from very little training data, as few as ten observations of the game state; and
- Demonstration of a model-based transfer learning approach, with little to no negative transfer, and with statistically significant improvement, across more than twenty games.

The methods explored in this thesis - object-oriented factorization, transfer ensembles, and model retraining - can help agents reduce generalization error in their predictive

models and maximize their performance. These tools open up possibilities for using much less data, noisier data, and higher-dimensional data, to train agents that continue to perform well in unfamiliar experiences.

Bibliography

[1] Ms pac-man competition, screen-capture, [online]. http://dces.essex.ac.uk/staff/sml/pacman/PacManContest.html, 2012.

[2] J. Asmuth and M. L. Littman. Learning is planning: Near bayes-optimal reinforcement learning via monte-carlo tree search. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*, 2011.

[3] C. G. Atkeson and J. C. Santamaria. A comparison of direct and model-based reinforcement learning. In *International Conference on Robotics and Automation*. Citeseer, 1997.

[4] S. Barrett, A. Rosenfeld, S. Kraus, and P. Stone. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence*, 242:132–171, 2017.

[5] J. Baxter. A model of inductive bias learning. *J. Artif. Intell. Res.(JAIR)*, 12:149–198, 2000.

[6] M. Bellemare, J. Veness, and M. Bowling. Bayesian learning of recursively factored environments. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1211–1219, 2013.

[7] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 06 2013.

[8] R. E. Bellman. *Adaptive control processes: a guided tour*, volume 4. Princeton university press Princeton, 1961.

[9] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[10] A. Braylan, M. Hollenbeck, E. Meyerson, and R. Miikkulainen. Reuse of neural modules for general video game playing. In *AAAI*, 2016.

[11] R. Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[12] W. Dai, Q. Yang, G.-R. Xue, and Y. Yu. Boosting for transfer learning. In *Proceedings of the 24th international conference on Machine learning*, pages 193–200. ACM, 2007.

[13] T. Degris, O. Sigaud, and P.-H. Wuillemin. Learning the structure of factored markov decision processes in reinforcement learning problems. In *Proceedings of the 23rd international conference on Machine learning*, pages 257–264. ACM, 2006.

[14] C. Diuk, A. Cohen, and M. L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 240–247. ACM, 2008.

[15] E. Eaton and M. DesJardins. Set-based boosting for instance-level transfer. In *Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on*, pages 422–428. IEEE, 2009.

[16] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius. Towards a video game description language. 2013.

[17] J. Gao, W. Fan, J. Jiang, and J. Han. Knowledge transfer via multiple model local structure mapping. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 283–291. ACM, 2008.

[18] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone. A neuroevolution approach to general atari game playing. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(4):355–366, 2014.

[19] M. Hausknecht and P. Stone. The impact of determinism on learning atari 2600 games. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[20] S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin. *Neural networks and learning machines*, volume 3. Pearson Education Upper Saddle River, 2009.

[21] T. Hester and P. Stone. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine learning*, 90(3):385–429, 2013.

[22] N. K. Jong and P. Stone. Model-based function approximation in reinforcement learning. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 95. ACM, 2007.

[23] J. Joseph, A. Geramifard, J. W. Roberts, J. P. How, and N. Roy. Reinforcement learning with misspecified model classes. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 939–946. IEEE, 2013.

[24] I. Millington and J. Funge. *Artificial intelligence for games*. CRC Press, 2009.

[25] T. M. Mitchell. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ., 1980.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[27] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pages 2845–2853, 2015.

[28] S. J. Pan and Q. Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

[29] P. Rohlfshagen and S. M. Lucas. Ms pac-man versus ghost team cec 2011 competition. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 70–77. IEEE, 2011.

[30] T. Schaul. A video game description language for model-based or interactive learning. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.

[31] J. Schrum, I. V. Karpov, and R. Miikkulainen. Human-like combat behaviour via multiobjective neuroevolution. In *Believable bots*, pages 119–150. Springer, 2013.

[32] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[33] M. E. Taylor, N. K. Jong, and P. Stone. Transferring instances for model-based reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 488–505. Springer, 2008.

[34] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[35] M. E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(Sep):2125–2167, 2007.

[36] S. Whiteson, B. Tanner, M. E. Taylor, and P. Stone. Generalized domains for empirical evaluations in reinforcement learning. In *In ICML 2009: proceedings of the twenty-sixth international*. Citeseer, 2009.